Nazir Nayal
62775

Final Project Report

QMBU 450
June 1, 2020

Intro:

In this project, I attempt to create a neural network model that predicts what rating from 1 to 5 does a food review text correspond to. That is, given a text which describes a customer's opinion on food services, the model predicts what would the rating this same person would have given for the service. The range of options for rating is integers from 1 to 5.

The usefulness of such a model is to create a numerical estimation of service quality using of text data. Numerical estimations make the quality date more interpretable.

Tools and Libraries:

For the implementation, python language is used. Here is a list of the libraries that need to be installed in order for the project to be run:

- numpy
- pandas
- cupy
- nltk
- spacy
- matplotlib
- collections
- gensim
- re
- bs4
- wordcloud
- torch
- random

For creating the neural network model, I have used PyTorch [7] framework. For word embeddings, GloVe [1] vectors of 300 dimensionality have been loaded and used to represent words. The IDE used to run the code is google colab notebooks, which provides a free gpu environment.

About Data:

The data used in the project is Amazon's fine food reviews [8], which contains data of food reviews that has been collected for a period of more than 10 years with up to around 500k reviews. The data contains several information about the reviews including product info, user ID, timestamps and several more. But the only two columns we are interested in for this project are the text review, and the score, which is the numerical rating given by the user.

Data Exploration:

First, here is the graph showing the distribution of classes in the dataset (Figure 1):

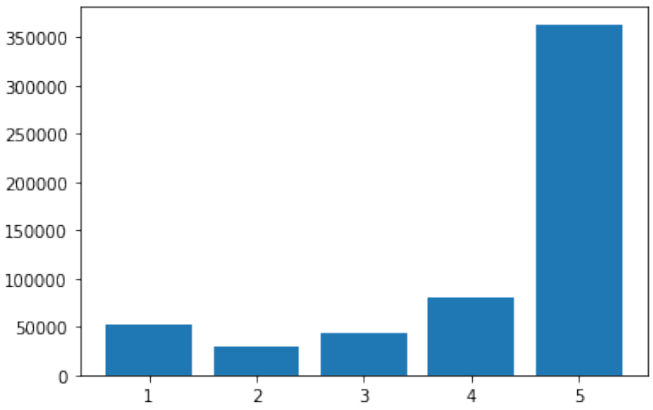


Figure 1: Class Distribution

It appears that the distribution is quite imbalanced and it mostly consists of class 5 instances, which will put restrictions on the number of instances used from each class in the training dataset in order to guarantee an equal number of instances for each class.

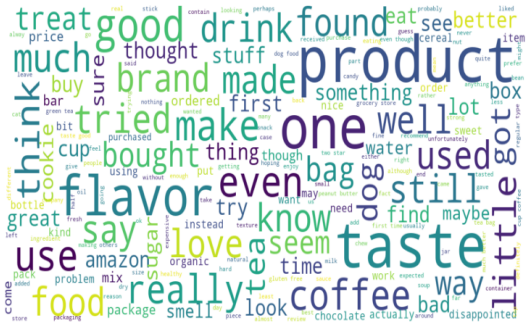
Then, for each class, I created a word cloud to get a glimpse of what kind of words are most common in each class. Each word cloud contains the top 200 frequent words (see Figure 2).

For these visualization tools, I have created a separate file names *TextVisualize.py*, which contains reusable helper functions for visualization.

In addition, some analysis on text features has been done and resulted in the following general stats:

- Average character count per review: 436.22
- Average word count per review: 80.26

(a) Class 1



(b) Class 2

(c) Class 3

(d) Class 4

(e) Class 5

Figure 2: Classes Word Cloud

- Total word count: 45626405

Data Preprocessing:

To process and clean the data, there have been several steps through which the data has been processed at in a pipelined manner. Like in the visualization case, a separate file named *TextPreprocess.py* has been created and contains several reusable utility functions that make preprocessing more modular and easier. This file is documented and added to the project files. Here are the steps taken in preprocessing the data in order of execution:

Lower Case

All textual data has been turned into lowercase, this is because subsequent steps assume all characters present are lowercase.

Expanding contractions

A map of handwritten contractions are added to the *TextPreprocess.py* file. This map is used to eliminate all possible contractions and replace those with plain format, this will make it easier for detecting the meanings of the words in the model.

Remove HTML Tags

A general scan just in case any tag exists.

Remove Multiple Spaces

If splitting based on white spaces is going to take place later for tokenization, multiple spaces would not be a problem, but in case any other method is used, this would keep the data in a safe format.

Remove Special Characters

Special Characters are defined to be any characters **other** than the following:

- alphabetic characters
- numerical digits

- white space
- - (dash)
- punctuation: ,.!?

Remove Punctuation

Punctuation characters being removed are [,.!?:;-=]. The reason punctuation was not considered among special characters is that in case we wanted to split a text to sentences, using these punctuation would be handy so leaving them for a later stage and then removing them would be possible.

Lemmatize Words

There were several options to use for lemmatizing words, two options that were considered are using *spacy* library, or using *WordNet* Lemmatizer provided by *nltk* [9] library. Finally I decided to use *WordNet* to lemmatize the words in the data.

Remove Stop Words

Removal is based the stop words list from the *nltk* [9] library.

Tokenize

Tokenization has been simply done by splitting the sentences with white space as delimeter.

Obtain Word Embeddings

Regarding word embeddings, there were 3 options in mind to use:

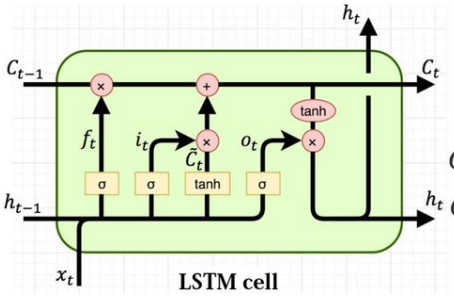
- Use *spaCy* [2] library vectorizer, which provides several models that can be found here
- Build a custom **Word2Vec** model using *gensim* [3] library
- Use pretrained *GloVe* [2] vectors.

The final runs of the code were done using GloVe [1] vectors. There are several versions of these vectors each with different dimensionality. The one chosen for this project is of dimensionality 300. After the vectors are loaded, the words in the data set are reduced to only those who have corresponding GloVe vectors. This reduces the size of the data set

which improves the speed of training, and leaves only the words that can be interpreted by the model. From the words, remaining, a word to index and its reverse (index to word) maps are created, in order to ease transition from text to embeddings for the model.

Model:

The model is essentially based on a variation of Recurrent Neural Networks (RNNs) called Long Short-term Memory (LSTM) [4]. Its implementation is available in several frameworks, like PyTorch [7] and TensorFlow [5], but in this project PyTorch [7] will be used to create the LSTM [4] model. Figure 3 shows the LSTM Cell structure as well as the equations used to calculate its weights and cell state variables.



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 3: LSTM Cell Structure [6]

The Model used in this project consists of the following layers in order from input to output:

Layers

Embedding Layer

Implemented using *nn.Embedding* provided by PyTorch. It is essentially a table of dimensionality $vocabularySize \times EmbeddingDimension$. In our case Embedding Dimension is 300, which is essentially the GloVe [1] vectors dimensionality. Normally, the embedding layer by PyTorch allows for training of the embeddings at the same time while model is training, but this require holding more weights and thus slows down the training. For our case, we have immediately loaded the pretrained vectors into the embedding layer.

LSTM Layer

the LSTM Layer is provided as *nn.LSTM* in PyTorch. It needs a minimum of two parameters to be created which are the size of the embeddings used, and the hidden state size. In a single run, the input to the LSTM is the output of the embedding layer in addition the hidden state from previous run. The hidden state is represented as a tuple of (hidden state, cell state).

Linear Layer

This layer takes the output of the LSTM and maps it to the outputs of the model, which should be of size equal to the number of classes we are attempting to classify. This layer is rather for the convenience of connecting the LSTM output to the Softmax activation layer, whose input should be equal to the number of classes.

Softmax Activation

The Softmax layer takes the input of the linear layer, and outputs the probabilities corresponding to each class. In the implementation log softmax is used instead of ordinary softmax.

Loss Function:

Loss function used for the model is Negative Log Likelihood, which is designed to take log softmax as input. The same effect could have been done by using a cross entropy loss function with a linear output only (without softmax).

Hyper-Parameters:

Here is a list of the hyper parameters we have for the model and their chosen values:

- Learning Rate: 0.0001
- Hidden State Size: 256
- Embedding Dimension: 300
- Vocabulary Size: 57366
- Epochs: 66

Training and Testing Data:

Class 2 has the least number of instances which is equal to 29744. For this, I have chosen 28k instance from each class to be in the training set. Then, we can use any number of the remaining instances for validation. for this I have chosen 80k instances for validation from all the remaining instances of all the classes.

Results:

After the model has been trained using the previously described parameters, it has yielded a 65 % accuracy on the validation set. Figure 4 below shows how the loss value has been changing while training.

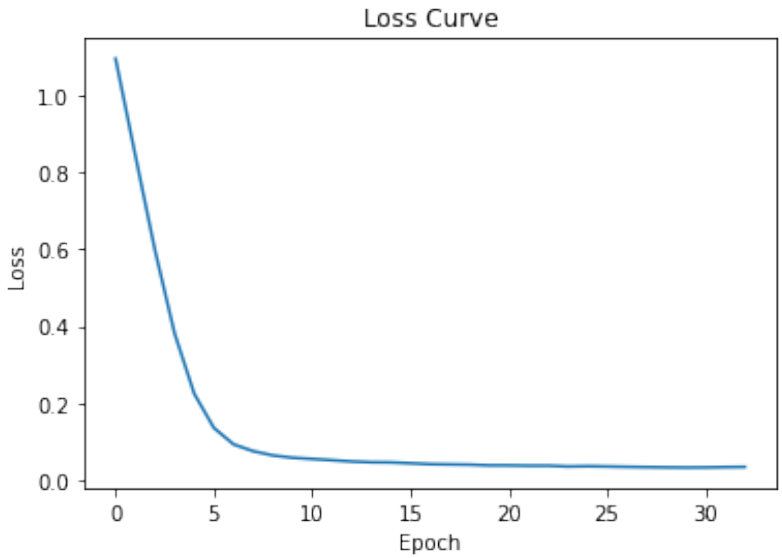


Figure 4: Loss Curve

Even though the final achieved accuracy is not as high as it should be, there are still many improvements that can be done on the model, but are left out due to limited time, and the fact that training takes a considerable amount of time (66 epochs took around 6 hours). Improvements that can be done may be parameter tuning, or possible further processing of the data.

Conclusion:

In total, I have create an LSTM Model that attempts to classify textual food reviews into 5 categories, where each category corresponds to a rating that is considered equivalent to the provided text. The model has been created using PyTorch [7] framework in python, and after several trials and testing of parameters, the best accuracy achieved was around 65 %.

References:

- [1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

- [2] spaCy
- [3] gensim
- [4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 15, 1997), 1735–1780. DOI:https://doi.org/10.1162/neco.1997.9.8.1735
- [5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [6] LSTM Figure
- [7] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d extquotesingle Alch'e-Buc, E. Fox, & R. Garnett (Eds.), Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
- [8] Amazon Fine Food Reviews Data
- [9] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.